REMARKS

Claims 1-19 are pending in the present application. No claims were canceled or added; claims 1, 3, 8, 13, and 18 were amended. Reconsideration of the claims is respectfully requested.

Support for the amendments to claims 1, 3, 8, 13, and 18 may be found in the Specification at least at page 2, lines 1 through 9, page 10, lines 21 through 27 and page 14, line 27 through page 19, line 15. No new matter has been added by these amendments.

An English translation of the abstract of reference JP10260820A, a Japanese patent application publication, is included with the present response. This was first identified in the IDS filed with the present application on July 24, 2003. Applicants request the Examiner to consider the reference.

I. 35 U.S.C. § 102, Anticipation

The Final Office Action dated May 31, 2007, rejects claims 1-19 under 35 U.S.C. § 102(b) as being anticipated by *Donohue et al.*, Method and a Mechanism for Synchronized Updating of Interoperating Software, U.S. Patent No. 6,202,207 (March 13, 2001) (hereinafter "*Donohue*"). This rejection is respectfully traversed.

I.A. Claims 1 and 2

The Final Office Action rejects claim 1 stating:

> Claims 1-19 are rejected under 35 U.S.C. 102(b) as being anticipated by *Donohue* (US 6,202,207 BI).
>
> Claim 1.
>
> Donohue further teaches a method for testing the compatibility of software versions (see at least FIGS.4A-B & associated text), the method comprising the computer implemented steps of:
>
> responsive to an installation of a new software module in a data processing system (see at least col.4:35-col.5:2), performing an inventory (see at least col.7:54-col.8:19) on an existing set of software modules resident in the data processing system (see at least 10, 80, 90 FIG.1 & associated text; 200, 210, 230, 290, 410 FIG.4A & associated text; col10:3-col.11:63);
>
> referring to a knowledge base of versions of respective software modules to obtain compatibility information for the new software module with the existing set of software modules (see at least 40 FIG. I & associated text; 110, 120, 130 FIG.2 & associated text; 250, 260 FIG.4A & associated text;col.7:54-col.8:19; col.9:35-64; col.11:45-62); and
>
> providing the compatibility information from the knowledge base, wherein the compatibility information is used to determine whether to install the new software module (see at least 310 FIG.4B & associated text; col.7:54-col.8:19; col.9:35-64; col.11:45-62).

Final Office Action dated May 31, 2007, pp. 6-7.

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.*, 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). In this case, each and every feature of the presently claimed invention is not identically shown in the cited reference, arranged as they are in the claims.

Amended independent claim 1 recites:

> 1. A method for testing the compatibility of software modules, the method comprising the computer implemented steps of:
>
> responsive to receiving a request to install a new software module in a data processing system, performing an inventory on an existing set of software modules resident in the data processing system;
>
> referring to a knowledge base of software modules to determine whether the new software module is known to function compatibly with the existing set of software modules to form compatibility information; and
>
> providing the compatibility information, wherein the compatibility information is used to determine whether to install the new software module.

In the present case, each and every step in claim 1 is not shown in the cited reference. In particular, *Donohue* fails to teach the feature of "referring to a knowledge base of software modules to determine whether the new software module is known to function compatibly with the existing set of software modules to form compatibility information." The Office Action cites *Donohue*, 40 FIG. I & associated text; 110, 120, 130 FIG.2 & associated text; 250, 260 FIG.4A & associated text; column 7, line 54 through column 8, line 19; column 9, lines 35 through 64, and column 11, lines 45 through 62, as allegedly teaching this feature.

> As shown in FIG. **1**, an updater component **20** is installed in system memory of a conventional network-connected computer system **10**, together with an associated computer program **30**. The updater component may have been delivered to the user of the local computer system on a storage medium (diskette or CD) for him to install, or it may have been explicitly downloaded from another computer system. In preferred embodiments of the invention, updater components are integrated within the computer program they are intended to maintain (or are otherwise delivered via the same mechanism and at the same time as their associated program). The updater component is then installed as part of the installation procedure of its associated program, such that the user is not required to take any special action to obtain or activate it. The installation of

each updater component includes the updater component registering itself with the operating system or another repository **40** on the local computer system. Thus, at least the updater components on the local system are identifiable and contactable by address information, and/or their product identifier, within the register entry. In alternative embodiments of the invention, the repository **40** may be a central or distributed repository for the network, as will be discussed below. It is a feature of the preferred embodiment of the invention that each updater component can locate, can be located by, and can communicate with other updater components which manage other software products. This capability is used when one updater component requires another one to update to a specific level before the former can execute its own update, as will be discussed below. This is enabled by the updater components registering within the operating system or other repository **40**. **[Donohue col. 7, line 54-col. 8, line 19]**

Updater components include data fields for an identifier and version number for their associated software products. The updater components may be delivered to customers with these fields set to null values, and then the installation procedure includes an initial step of the updater component interrogating its software product to obtain an identifier and current program version and release number. Alternatively, the software vendor may pre-code the relevant product ID and version number into the updater component. The system **10** of FIG. **1** is shown connected within a network 100 of computers including a number of remote server systems (**50**,**50'**) from which software resources are available for applying updates to programs installed on the local system **10**. Each server system includes within storage a list **60** of the latest versions of, and patches for, software products which are available from that server. Each vendor is assumed here to make available via their Web sites such a list **60** of software updates (an example of which is shown in FIG. **2**) comprising their product release history, in a format which is readable by updater components, and to make available the software resources **70** required to build the releases from a given level to a new level (this transition from a software product release to a new level will be referred to hereafter as a `growth` path). The entries in the software updates list **60** include for each software product version **110** an identification **120** of the software resources required for applying the update and an identification **130** of its prerequisite software products and their version numbers. **[col. 9, lines 36-64]**

The updater component performs **290** (see FIGS. **3** and **4**) a scan of the operating system file system to check whether the required software resources are already available on the local computer system. The required resources are the software update artifacts required to bring the current application software to the new level, and the software updates required for updating prerequisite software to required levels. Each updater component associated with pre-requisite installed products is contacted **300** to ensure (a) that it is installed on the local system, and (b) that it is at or greater than the version level required for interoperability. If all required resources are available locally (or on another machine in the case of software relying on some pre-requisite software operating on a remote machine), and have been verified, then the updater component progresses to the step **310** (see FIG. **4**) of building the updated software version. **[col. 11, lines 46-62]**

Donohue, in column 7, line 54 through column 8, line 19, teaches that an updater component is installed on a system for each software module on the system and that this updater registers with the operating system or some other repository. *Donohue*, in column 9, lines 35 through 64, teaches that the updater component has fields for an associated identifier and version number. Additionally, servers in a network contain the latest versions of, and patches for, software available on the server. The entries for each software product version includes an identification of the software resources required for applying the update and an identification of its prerequisite software and their version numbers. *Donohue*, in column 11, lines 45 through 62, teaches that the updater component scans the operating system file system to check whether the required software resources are already available on the local computer system. Further, each updater associated with pre-requisite installed products is contacted to ensure that it is installed on the local system and that it is at, or greater than, the version level required for installation.

Thus, the passages of *Donohue*, column 7, line 54 through column 8, line 19, column 9, lines 35 through 64, and column 11, lines 45 through 62, whether taken separately or read together, fail to teach the feature of "referring to a knowledge base of software modules to determine whether the new software module is known to function compatibly with the existing set of software modules to form compatibility information." Rather, the cited passages of *Donohue* teach that the updater component of *Donohue* merely checks to determine whether prerequisite software is available on a system before installing an update or new software. If the perquisites are not available, the update component then causes the prerequisites to be installed. In contrast, claim 1 recites determining whether the new software module is known to function compatibly with the existing set of software modules.

Donohue fails to teach making a determination as to whether the new software functions compatibly with other software modules. *Donohue* teaches ensuring that prerequisite software is installed and updating the current version. However, as taught by *Donohue*, a user could receive an update to a spreadsheet program on their system. The updater of *Donohue* would merely determine if the prerequisite software for updating the spreadsheet program is installed, and if the perquisites were installed, updating the spreadsheet program. However, the updated spreadsheet program may now produce spreadsheets that cannot be used by other software modules on the system, such as a word processor, or the e-mail client, and so forth. Furthermore, updating the spreadsheet program may now cause conflicts with other programs by updating a shared component, which the other software module can now no longer use because it has been updated for the spreadsheet program. Therefore, the new software module does not function

compatibly with other software modules on the local data processing system. Thus, *Donohue* fails to teach determining whether the new software functions compatibly with other software modules to form compatibility information.

Furthermore, *Donohue* does not teach determining if the new software module is known to function compatibly with the existing set of software modules. *Donohue* does not teach determining compatibility with the set of software on the local data processing system. Rather, *Donohue* teaches determining if the new version of the software module is compatible only with a sub-set of the software modules. That is, *Donohue* only teaches determining if the software that makes up the prerequisites for the new version of the software is installed. *Donohue* does not teach checking to see if the new software is compatible with any other software on the local data processing system other than the prerequisite software. Thus, *Donohue* does not teach making the determination for the existing set of software modules resident in the data processing system.

Additionally, *Donohue* does not teach referring to a knowledge base of software modules. The repository taught by *Donohue* is a repository of updater paths and network addresses. (see *Donohue*, column 8, lines 19 through 30.) No where does *Donohue* teach that the repository taught by *Donohue* is a knowledge base of software modules. Rather, each updater keeps track of the current version of the associated software and a list of prerequisite software. A repository of one set of information cannot teach a repository of another et of information for the purposes of anticipation under 35 U.S.C. § 102(b). Therefore, a repository containing information about the path to a plurality of updaters does not teach a knowledge base of software modules. Thus, *Donohue*'s teaching of referring to a repository of updater paths and network addresses does not teach the feature of "referring to a knowledge base of software modules to determine whether the new software module is known to function compatibly with the existing set of software modules to form compatibility information," as recited in claim 1

Therefore, *Donohue* fails to teach the feature of "referring to a knowledge base of software modules to determine whether the new software module is known to function compatibly with the existing set of software modules to form compatibility information."

Additionally, *Donohue* fails to teach the feature of "providing the compatibility information, wherein the compatibility information is used to determine whether to install the new software module." As argued above, *Donohue* does not teach compatibility information as recited in claim 1. Therefore, logically, *Donohue* cannot teach providing the compatibility information, or using the compatibility information to determine whether to install the new

software module.

Further, *Donohue* does not teach determining whether to install the new software module. Rather, as explained above, *Donohue* teaches determining if prerequisite software is currently installed on a local data processing system. If the perquisites are installed already, then the new software is installed. If the perquisites are not installed, then the updater of *Donohue* causes whichever software is missing to be installed, and then the updater installs the new software. Thus, as taught by *Donohue*, the new software is simply installed, and there is no determination as to whether to install the new software.

Thus, for at least the reasons set forth above, *Donohue* fails to teach amended independent claim 1. Consequently, dependent claim 2 is not anticipated at least by virtue of depending from independent claim 1.

I.B. Claims 3-5, 8-10, 13-15, and 18

Amended independent claim 3, which is representative of amended independent claims 8, 13, and 18, recites:

> 3. A method for testing the compatibility of software modules, the method comprising the computer implemented steps of:
>
> responsive to receiving a request to install a new software module in a data processing system, performing an inventory on an existing set of software modules resident in the data processing system;
>
> referring to a knowledge base of software modules to determine whether the new software module is known to function compatibly with the existing set of software modules; and
>
> responsive to a negative determination, testing the new software module in a test data processing system in combination with the existing set of software modules.

Amended independent claim 3 recites language similar to that of amended independent claim 1. As argued above in regards to amended independent claim 1, *Donohue* fails to teach the feature of "referring to a knowledge base of software modules to determine whether the new software module is known to function compatibly with the existing set of software modules."

Additionally, *Donohue* fails to teach the feature of "responsive to a negative determination, testing the new software module in a test data processing system in combination with the existing set of software modules." As *Donohue* fails to teach making a determination as to whether the new software module is known to function compatibly with the existing set of software modules; logically, *Donohue* cannot teach testing the new software module in a test data processing system in combination with the existing set of software modules in response to a negative determination in regards to whether the new software module is known to function

compatibly with the existing set of software modules.

Furthermore, *Donohue* does not teach testing a new software module. The Office Action cites to element 290 of Figure 4A and associated text as allegedly teaching this feature. However, as stated in Figure 4A and column 11, lines 45 through 50, Element 290 merely teaches scanning the local data processing system to determine if the prerequisites for the software are installed. Neither Figure 4A, nor this passage, nor any other passage of *Donohue* teaches anything about testing the new software module in a test data processing system in combination with the existing set of software modules. *Donohue* teaches an automatic updater that also automatically causes prerequisite programs to be updated as well. *Donohue* does not teach testing software combinations. *Donohue* also does not teach or mention a test data processing system. *Donohue* simply teaches installing the new software on the local data processing system.

Therefore, for at least the reasons set forth above, *Donohue* does not anticipate amended independent claim 3 under 35 U.S.C. § 102(b). Furthermore, as claim 3 is representative of claims 8, 13, and 18, the distinctions between claim 3 and *Donohue* apply to claims 8, 13, and 18. Consequently, dependent claims 4-5, 9-10, 14-15, and 19-20 are also allowable at least by virtue of their dependence from one of these independent claims.

I.C. Claims 6, 7, 11, 12, 16, 17, and 19

Independent claim 6, which is representative of independent claims 11, 16, and 19, recites:

> 6. A method in a data processing system for monitoring software combinations, the method comprising:
> identifying a software module;
> determining whether information is present for the software module;
> if information is absent for the software module, installing the software module to form an installed software module; and
> testing the installed software module.

Donohue fails to teach the feature of "testing the installed software module." As argued above in regards to amended independent claim 3, *Donohue* does not teach testing software. Therefore, for the reasons described above, *Donohue* fails to anticipate independent claim 6 under 35 U.S.C. § 102(b). Furthermore, as claim 6 is representative of claims 11, 16, and 19, the distinctions between claim 3 and *Donohue* apply to claims 11, 16, and 19. Consequently, dependent claims 7, 12, and 17 are also allowable at least by virtue of their dependence from one of these independent claims.

Therefore, the rejection of claims 1-19 under 35 U.S.C. § 102(b) has been overcome.

II. Conclusion

It is respectfully urged that the subject application is patentable over *Donohue* and is now in condition for allowance.

The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: August 31, 2007

Respectfully submitted,

/Gerald H. Glanzman/

Gerald H. Glanzman
Reg. No. 25,035
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicants

GHG/blj